Merging on-demand HPC resources from Amazon EC2 with the grid: a case study of a Xmipp application

Alejandro Lorca[‡12], Javier Martín-Caro[§2], Rafael Núñez-Ramírez[¶3] and Javier Martínez-Salazar[‖3]

[1] Instituto de Física de Cantabria. CSIC
[2] Secretaría General Adjunta de Informática. CSIC
[3] Departamento de Física Macromolecular, Instituto de Estructura de la Materia. CSIC

Abstract. We present an infrastructure in which HPC resources from the Amazon Web Services public cloud are combined with specific grid resources at Ibergrid. The integration is done transparently for the GridWay users through a daemon which permanently monitors the pool of available resources and submitted jobs, managing virtual instances for satisfying the demand under budget restrictions. The study has been proved with a specific application from the Xmipp package, which performs image processing from electron microscopy data and requires heavy high-throughput computing offering parallelization capabilities. The application was ported successfully for such a hybrid framework with the help of the MPI-Start package. Some preliminary results from test runs are presented for a controlled sample of thousand input images. Some inconveniences and troublesome aspects of the deployment are also reported.

1 Introduction

There is common agreement regarding the status reached by cloud computing being on top of the peak of expectations according to the Gartner's hype cycle of technologies [1]. This differs from what has already happened to grid computing, which is getting closer to a productivity plateau as required by the Large Hadron Collider experiments and supported by other scientific communities.

Simultaneously, the development achieved by the architecture of the processor has brought to the market multi-core servers at very affordable prices even for small-to-medium research units. Not only the speed, but the parallelization, virtualization and energy-saving capabilities of the hardware offer compelling reasons to upgrade the equipment.

The resources offered by the institutions taking part in grid computing are slowly moving into this direction. Looking at the figures offered by the Ibergrid

[‡] e-mail of corresponding author: alejandro.lorca@cti.csic.es
[§] e-mail: javier.mcaro@orgc.csic.es
[¶] e-mail: rafa@iem.cfmac.csic.es
[‖] e-mail: jmsalazar@iem.cfmac.csic.es

infrastructure in February 2011 [2], 12,257 declared logical CPUs (cores) are sustained by 3,359 physical CPUs (processors) averaging 3.65 cores/processor, far above the single-core architecture. Similar increases have occurred regarding memory and network speed since the early days of grid computing.

Therefore, it seems natural to port to the grid high performance computing (HPC) applications or, at least, the so-called many task computing (MTC) ones. Merging resources coming from the cloud would also appear to be a step forward in introducing an important degree of flexibility into a relatively rigid infrastructure. To this end, the key role of Amazon EC2 [1] and other providers of Infrastructure as a Service utility model have made affordable the use of on-demand virtual instances for the general public. A service based on a static infrastructure which can be enlarged and reduced dynamically is still a challenge and there are remarkable European ongoing projects appointed on the subject, focusing on site provisioning[2], virtual environments[3] and e-infrastructure enrichment[4].

In this paper we present a simple model of hybrid infrastructure grid-cloud in Sect. 2, with emphasis in a transparent experience for the user. An application used for molecular imaging is described in Sect. 3 since it has been used whilst developing the framework. The porting of the application to the hybrid infrastructure is detailed in Sect. 4. Some preliminary results are shown in Sect. 5 for both, technical and scientific interest with comments on some issues which appear during the implementation of the infrastructure. We summarize the study in Sect. 6 with some recommendations and future work.

2 The grid-cloud model

2.1 Architecture

In our model, an heterogeneous grid composed out Ibergrid nodes is considered under the following restrictions:

1. gLite 3.1 lcg-type computing elements (CEs),
2. support of any of the Ibergrid VOs or CSIC VOs,
3. capability of handling MPI jobs.

The available CEs are shown in Table 1.

Additionally, we consider many particular High CPU Extra Large instances of Amazon EC2, described in Table 2. The virtual machines are launched according to a customized Amazon Machine Image (AMI) and are capable of handling MPI jobs through the Open MPI suite and secure-shell. The AMI has been saved in an Amazon S3 bucket during the usage of the infrastructure.

The last element required in order to submit jobs is the user interface. It is a dedicated machine physically located at the institute SGAI-CSIC and accessible to the users.

[1] `http://aws.amazon.com/ec2`
[2] Stratuslab. `http://www.stratuslab.eu`
[3] Venus-C. `http://www.venus-c.eu`
[4] Siena Inititative. `http://www.sienainitiative.eu`

ARCH (bits)	CPU (MHz)	MEM (MB)	Cores	LRMS	Endpoint	Site
64	2400	4058	36	lcgsge	ce01.up.pt	UPorto
64	3200	15636	1456	sge	egeece01.ifca.es	IFCA-LCG2
64	3200	15636	1456	sge	egeece02.ifca.es	IFCA-LCG2
64	3200	15636	1456	sge	egeece03.ifca.es	IFCA-LCG2
64	3200	2048	416	lcgpbs	ce.iaa.csic.es	IAA-CSIC
32	2330	512	12	lcgpbs	ce.cp.di.uminho.pt	UMinho-CP

Table 1. Resources from LCG-CEs in Ibergrid with MPI support.

ARCH (bits)	CPU (EC2 CU)	MEM (MB)	Cores	LRMS	API name	Region	Price ($/hour)
64	2.50	7000	8	none	c1.xlarge	US-EAST	0.68

Table 2. Resource type used from Amazon EC2.

2.2 Middleware

The user interface . It hosts the middleware handling the submission of jobs, and thus needs to provide the different services which are required by the lcg-CE computing elements: creation of proxies, verification from a VO, file transfer mechanism, command execution, etc. An installation of the gLite User Interface[5] and the Globus Toolkit[6] suffices for these purposes.

On top of that, a job scheduler different from the gLite WMS is required, since the power coming from the cloud has to be locally managed. The GridWay metascheduler [3] is a tool designed for the submission, scheduling, control and monitor of jobs from a single access point. In the last available version[7] it includes a ssh plugin, gaining remote access to cloud instances.

The GW_EC2 service manager. A deeper analysis on the many grid-cloud enabling mechanisms has been discussed elsewhere [4], proposing also a general framework based on GridWay where a "Service Manager" would take care of the interoperability with cloud resources. So far we have no evidence of any implementation of such service, being our contribution a novel attempt which validates the framework. It consists of four modules:

1. Amazon Web Services account, where the system administrator (GridWay administrator) has access to in order to launch instances and be subsequently billed. The module can read the certificates off for issuing such operations.
2. Budget policy, regarding the limits to start and finish instances. Here we propose a simple budget rate scheme where the rate limit is input in terms of amount of money per time unit.

[5] gLite UI v3.2.8-0. `http://glite.cern.ch/glite-UI/`
[6] Globus Toolkit v4.0.8. `http://www.globus.org/toolkit/downloads/4.0.8/`
[7] GridWay v5.6.1. `http://gridway.org`, `http://dev.gridway.org`

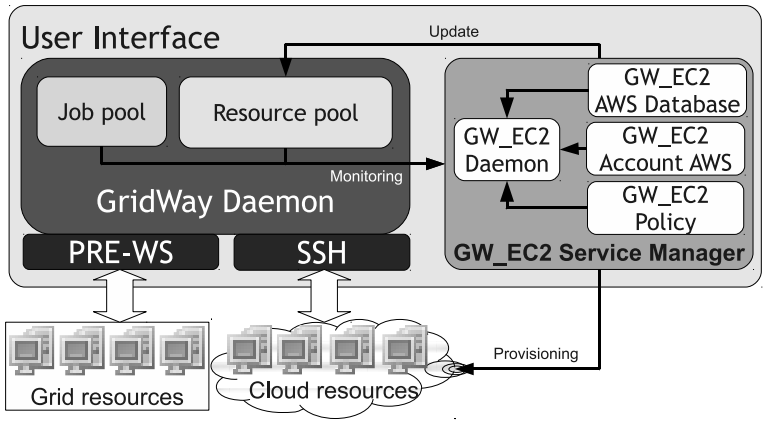

Fig. 1. Scheme of the middleware design. The enclosing light-gray box represents the user interface for job submission where the local components are installed. The left solid block stands for the GridWay package situated above the execution plugins to interact with the resources. The novel contribution (GW_EC2) is placed on the right showing the four existing modules.

3. Provider database, making the system is aware of the different machine types offered by the provider and its pricing. This information could be properly updated at running time.
4. A daemon, monitoring both the job and the resource pools from GridWay and communicating with the provider. According to some configuration parameters and the other modules, it decides when the pending jobs deserve dedicated machines for their execution and how many to launch. It does also the opposite action; to keep an eye if the job queue gets empty and shutdown the instances. The daemon uses the Amazon EC2 API Tools[8] for this purpose.

The interaction of the different components is depicted in Fig. 1.

2.3 HPC-enabler

Due to the heterogeneous environment found in the grid, the necessity of a single interface to process parallel jobs arises. MPI-Start [5] proposes an interface to hide the implementation details for the submitted jobs. It is composed by a set of scripts which process all the life cycle of the job, including MPI implementation and the additional features needed before and after the parallel execution. Because it can be installed also on single machines without a local resource manager, the implementation favors the usage of the Amazon EC2 multi-core instances.

For the study we prepared a private AMI based on the Amazon Linux AMI, with Open MPI and MPI-Start. A specific set of users were created on it with passwordless ssh-access from the user interface through public keys.

[8] `http://aws.amazon.com/developertools/351`.

3 Scientific usage

3.1 Specific problem

As an example of a scientific problem which required high computational resources we have used image processing of electron microscopy images of biological macromolecules. This technique allows the visualization and characterization of the structure of large macromolecular complexes. Due to the requirement of a low electron dose to minimize radiation damage during image recording the images typically suffer from low signal to noise ratio.

In the last few years several image processing algorithms have been developed in order to improve the signal to noise ratio of these images [6]. Basically, these algorithms consist of 2D translational and rotational alignments of the images which yield an average image with an improved signal to noise ratio. Such averaging is only possible if the experimental images correspond to the same orientation of the same macromolecule. However, it is very common for electron microscopy data set to contain more than one different 2D structure. For that reason several classification methods have been developed which curiously require that the images are aligned beforehand. This paradoxical situation has been solved with the developing of algorithms which combine 2D alignment with classification iteratively, known as multi-reference alignment. The outputs of these algorithms are averaged images with improved signal to noise ratio for each of the subgroups in the original data set (Fig. 2).

3.2 Xmipp

The Xmipp package is a suite of electron microscopy image processing programs [7]. Among other, Xmipp include an algorithm for 2D alignment and classification widely used by the electron microscopy community, named as maximum likelihood multi-reference refinement (ML2D). The details and mathematical background of ML2D are explained in [8].

Briefly, the set of images are compared to a predefined number of reference images, which are assumed to represent the structural diversity of the data. Each experimental image is compared and aligned with respect to all references and a probability is assigned to each combination of alignment and reference. Since each experimental image is compared to all references in all possible rotations and translations the searching space became huge. This makes ML2D computationally expensive and high performance computing is advantageous.

In this work we have tested the availability of grid computing for ML2D. We have used as experimental data electron microscopy images of GroEL, a large macromolecular complex from *Escherichia coli bacteria*.

4 Application porting

The aforementioned Xmipp application was ported from cluster computing to the grid-cloud infrastructure by creating a self-contained pack of scripts, input files and

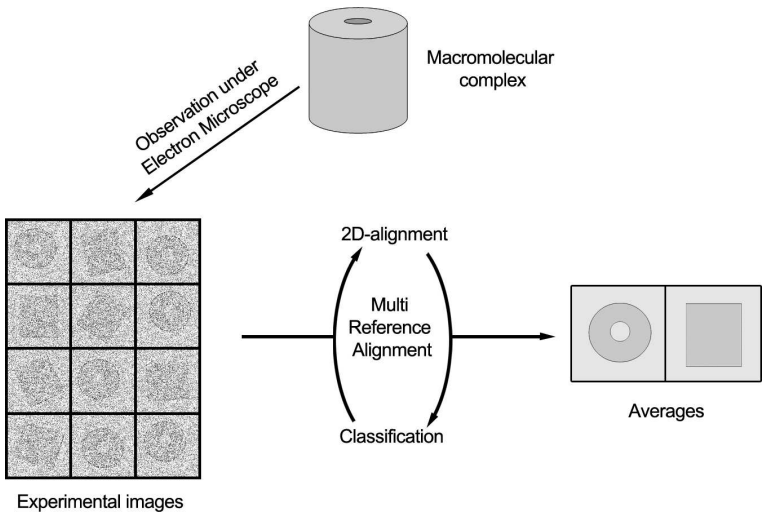

Fig. 2. Schematic representation of multi-reference alignment procedure. The 3D structure of the macromolecular complex is observed under the electron microscope as very noisy 2D images. Thousands of these images are processed by multi-reference alignment procedures to obtain average images with improved signal to noise ratio.

the Xmipp package. The strategy was to perform a dedicated remote compilation of the xmipp_mpi_ml_align2d program using, whenever possible, as many parallel threads as were asked for. The user submits a template like this:

```
NAME=ml2d_1000_8
EXECUTABLE=mpi-start-wrapper.sh
ARGUMENTS=ml2d OPENMPI 8
TYPE="single"
NP=8
INPUT_FILES=mpi-start-wrapper.sh, mpi-hooks.sh, ml2d, images-1000.tgz, Xmipp-2.4-src.tar.gz
OUTPUT_FILES=ml2d.tgz
STDOUT_FILE=ml2d.out
STDERR_FILE=ml2d.err
```

altogether with a set of files:

- Two mpi scripts: the first one is the wrapper executable and it is almost the same for every job (mpi-start-wrapper.sh) accepting as arguments the file to run in parallel (ml2d), the MPI taste to use (OPENMPI) and the parallelization degree for the case of ssh execution (8), otherwise the NP parameter will be used. The other script (mpi-hooks.sh) deals with additional features before and after the MPI execution.
- The script for run (ml2d) sets the library paths and calls the application with the configuration parameters.
- The compressed files, including the input images (images-1000.tgz) for processing and the package distribution (Xmipp-2.4-src.tar.gz).

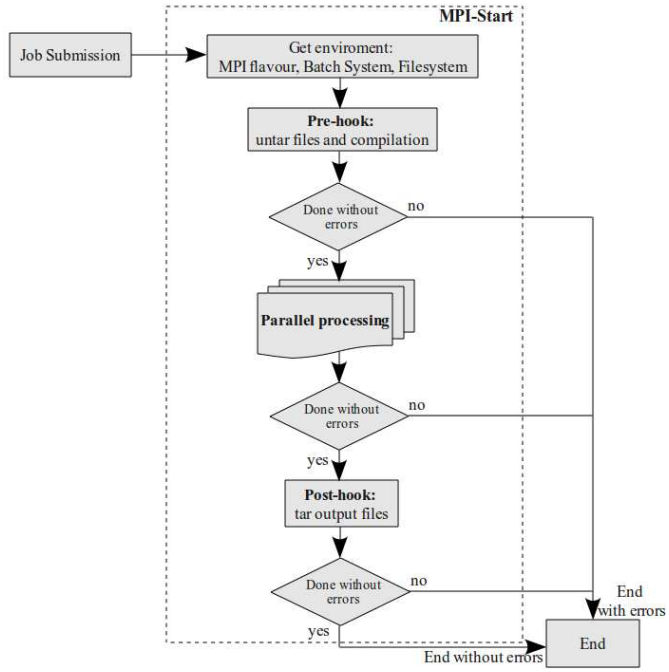

Fig. 3. The MPI-Start plays a wrapper role between the middleware and hardware stacks.

The submission and job control in then handled by GridWay automatically, according to its own configuration. When the job is submitted and accepted by a resource, it is when the MPI-Start takes care of the correct settings, prepares the execution in the "pre-hook" phase and, after the parallel processing, does a similar "post-hook" action in order to retrieve the output. The flow control is sketched in Fig. 3.

5 Results and discussion

5.1 Application profile

The application was run in many occasions, obtaining more averaged images and testing the underlying infrastructure. A very nice feature observed was the scaling shown for the speed-up due to parallelization. In Fig. 4 equivalent test runs for the application were considered, taking 1000 input images and 4 reference output images. In comparison to the single-core run, the 2-cores run takes 54% of time, which gets quite close to the theoretical half-time limit. Equally the 4-cores corresponds to a 57% of the 2-cores time and the 8-cores job takes 57% of the time employed by 4-cores task. We observed an standard deviation for the total time taken by the same jobs of about 10%. Note that also the pre-execution time gets reduced due to

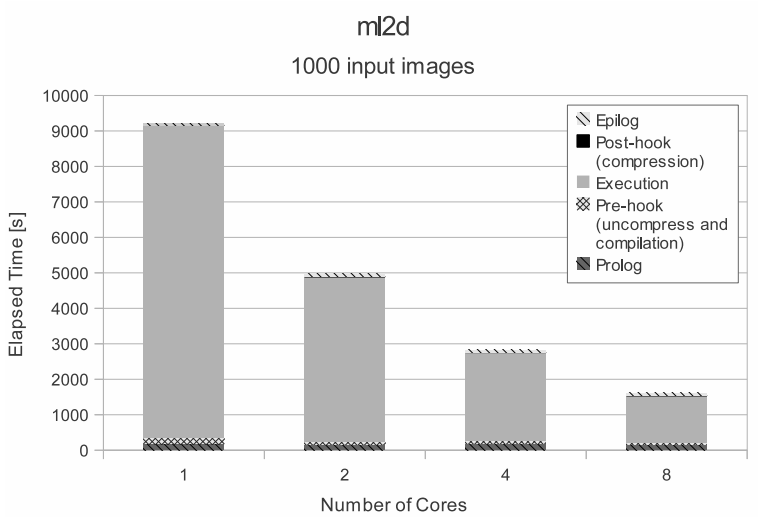

Fig. 4. Scaling of the application for an input sample of 1000 images averaged to 4 reference output images. The columns indicate the total elapsed time for each job, consisting of input transfer (prolog), pre-execution (pre-hooks), execution, post-execution (post-hooks) and output transfer (epilog). The four jobs were exclusively run in the same Amazon instance at different times.

the multi-threaded option at compile for the Xmipp package. Runs at the lcg-CEs showed a much larger deviation (20%) with high failure rate, depending on the hardware and cluster availability. They also suffer from the undesirable waiting time for free slots, but in general one could expect jobs being done at IFCA-CSIC on a bit more time than the invested in Amazon EC2. For the other sites we were unable to run successfully the jobs. A more detailed summary of job runs is given in Table 3.

During execution, all the cores were used to a very large degree consuming all the available CPU cycles. On the contrary, memory was not a limitation for the kind of hardware tested. Access to disk became relevant uncompressing the package at the pre-execution stage, lowering the performance of the application in those systems with slow I/O access. Networked shared file-systems without low-latency, showed this performance degradation as well, added up to the MPI communication layer saturation.

5.2 Daemon behavior

The GW_EC2 being still experimental delivered a very smooth behavior. It monitors the pool of jobs and resources known to GridWay at each configuration step time (one minute by default). From there on, if there are pending jobs in the queue, it launches a determined amount of Amazon instances in order to satisfy the jobs' conditions. The daemon stops launching new instances if there are no more jobs

Job	SITE	Cores	Prolog	Pre-hook	Execution	Post-hook	Epilog	Total
					[hh:mm:ss]			
ml2d_1000_1	AWS EC2	1	2:46	3:13	2:26:38	0:01	1:01	2:33:49
ml2d_1000_2	AWS EC2	2	2:30	1:54	1:17:26	0:01	1:42	1:23:33
ml2d_1000_4	AWS EC2	4	3:01	1:26	0:41:25	0:01	1:15	0:47:08
ml2d_1000_8	AWS EC2	8	2:39	0:57	0:22:07	0:01	1:15	0:26:59
ml2d_1000_4	IFCA-CSIC	4	0:04	3:30	0:48:41	0:05	0:02	0:52:22
ml2d_1000_8	IFCA-CSIC	8	0:04	4:57	0:29:26	0:05	0:02	0:34:34
ml2d_1000_16	IFCA-CSIC	16	0:04	3:33	0:16:11	0:02	0:02	0:19:52

Table 3. Detailed elapsed time at the different phases for each job. For the IFCA-CSIC the pending time on the remote queue has not been taken into account.

waiting on the local system or if the policy budget rate has been reached, leading to a maximum amount of enrolled machines. We deployed an infrastructure with a running costs limited to $10/h, using at most 14 simultaneous instances as shown in Table 4.

The enrollment of the instances into the GridWay pool of resources happens as soon as the GridWay daemon is scheduled to do a discovery of new hosts. Because this is a costly action, the daemon realizes about new resources every `DISCOVERY_INTERVAL` and we set that value to 60s. When there are no more jobs in a given cloud resource nor pending in the local queue, then the instance is marked for shutdown but kept as long as it approaches the billed hour, just for the case of new entering jobs. The daemon keep a historic log of the machines but currently cost savings are only spared and not used at a later moment.

```
HID OS            ARCH  MHZ %CPU MEM(F/T  )  DISK(F/T)       N(U/F/T)    LRMS             HOSTNAME
 0 LinuxUnknown   x86_6 2667 100   7000/7000 1690000/1690000       4/8/8 jobmanager-ssh   ec2-50-16-72-188.compute-1.amazonaws.com
 1 LinuxUnknown   x86_6 2667 100   7000/7000 1690000/1690000       4/8/8 jobmanager-ssh   ec2-184-72-161-10.compute-1.amazonaws.com
 2 LinuxUnknown   x86_6 2667 100   7000/7000 1690000/1690000       8/8/8 jobmanager-ssh   ec2-50-16-182-233.compute-1.amazonaws.com
 3 ScientificCERNS x86_6 3200   0 15636/15636            0/0 0/1456/1456 jobmanager-sge   egeece01.ifca.es
 4 ScientificCERNS x86_6 3200   0 15636/15636            0/0 7/1456/1456 jobmanager-sge   egeece03.ifca.es
 5 ScientificCERNS i386  2330   0     512/512            0/0     0/12/12 jobmanager-lcgpbs ce.cp.di.uminho.pt
 6 LinuxUnknown   x86_6 2667 100   7000/7000 1690000/1690000       4/8/8 jobmanager-ssh   ec2-174-129-180-84.compute-1.amazonaws.com
 7 LinuxUnknown   x86_6 2667 100   7000/7000 1690000/1690000       8/8/8 jobmanager-ssh   ec2-174-129-78-66.compute-1.amazonaws.com
 8 ScientificCERNS x86_6 3200   0   2048/2048            0/0   3/406/416 jobmanager-lcgpbs ce.iaa.csic.es
 9 LinuxUnknown   x86_6 2667 100   7000/7000 1690000/1690000       8/8/8 jobmanager-ssh   ec2-50-17-116-131.compute-1.amazonaws.com
10 LinuxUnknown   x86_6 2667 100   7000/7000 1690000/1690000       8/8/8 jobmanager-ssh   ec2-75-101-219-12.compute-1.amazonaws.com
11 LinuxUnknown   x86_6 2667 100   7000/7000 1690000/1690000       8/8/8 jobmanager-ssh   ec2-50-16-128-24.compute-1.amazonaws.com
12 LinuxUnknown   x86_6 2667 100   7000/7000 1690000/1690000       8/8/8 jobmanager-ssh   ec2-174-129-114-17.compute-1.amazonaws.com
13 LinuxUnknown   x86_6 2667 100   7000/7000 1690000/1690000       8/8/8 jobmanager-ssh   ec2-184-72-169-195.compute-1.amazonaws.com
14 ScientificSLSL x86_6 2400   0   4058/4058            0/0     0/36/36 jobmanager-lcgsge ce01.up.pt
15 LinuxUnknown   x86_6 2667 100   7000/7000 1690000/1690000       8/8/8 jobmanager-ssh   ec2-174-129-60-28.compute-1.amazonaws.com
16 LinuxUnknown   x86_6 2667 100   7000/7000 1690000/1690000       4/8/8 jobmanager-ssh   ec2-184-73-25-138.compute-1.amazonaws.com
17 LinuxUnknown   x86_6 2667 100   7000/7000 1690000/1690000       8/8/8 jobmanager-ssh   ec2-67-202-16-53.compute-1.amazonaws.com
18 LinuxUnknown   x86_6 2667 100   7000/7000 1690000/1690000       8/8/8 jobmanager-ssh   ec2-50-16-115-39.compute-1.amazonaws.com
```

Table 4. Output of the `gwhost` command from the user interface showing the hybrid grid-cloud resources. A set of 14 Amazon instances are enrolled in the pool with the hosts of the lcg-CEs with MPI-Support in Ibergrid.

5.3 Image classification and averaging

After the executions of ML2D algorithm in the grid the output data are transferred to a local machine for further analysis. The algorithm produces averaged images and "doc" files for each iteration. The doc files provide detailed information about the classification and alignment process, for example, the proportion of experimental images belonging to each average, the rotations and translation of each image to be aligned, etc. The averaged images must be visualized with specific graphical applications available in the Xmipp package.

Fig. 5 shows an example of these results. Whereas the original experimental images are noisy and featureless, the ML2D output averages display an increased signal to noise ratio which allows the description of structural details of the macromolecular complex. Several parameters of the macromolecule as size, shape, geometry, number of subunit, etc. can be inferred from the averaged images. This information is extremely useful for molecular biologists since the structure and function are parameters strongly related in biological macromolecules. Thus, the analysis of the averaged images of a macromolecular complex could shed light on how this complex works. In our example, the ML2D classification and alignment of 1000 experimental images yield two types of averaged images, one of them circular and with seven main masses and the other one square and with four parallel strips. Based in this information we are able to state that GroEL is a macromolecule organized as a cylinder with a seven-fold symmetry axis as has been observed in numerous previous work.

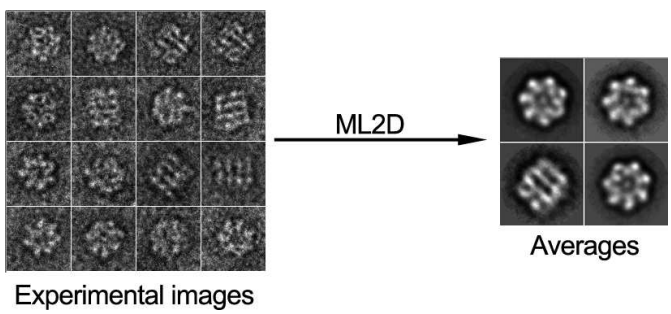

Fig. 5. ML2D classification of 1000 electron microscopy experimental images into 4 final improved images after a typical run of the application.

5.4 Troubleshooting

During the deployment of the research infrastructure, some technical issues arose difficulting the successful interoperability of the grid-cloud platform. Let us summarize the most relevant aspects:

- **GridWay ssh MAD**. We had some troubles when using it under a multi-user environment. Some log files were written by one user into `/tmp/ssh.log` and `/tmp/ssh_tm.log` but they were unable to be overwritten by another users, rendering the MAD useful only for a single user.
- **GRAM2 and MPI-Start versioning at sites**. Because each site deploys a different version, some of the jobs described through a rsl job submission format used by GridWay did not run properly. It happened than in IAA-CSIC and UMinho-CP sites, instead of receive NP processors, a single processor was given, narrowing the availability of resources. Curiously enough, those sites did not show any problem whatsoever to run parallel jobs with the same version of MPI-Start used and the gLite JDL.
- **Firewall**. The fact that the CSIC uses a firewall for controlling the access and the behavior of net traffic makes distributed computing more difficult. We dealt with unexpected time-outs due to inactive sessions which were still waiting for updates on job status, causing frequent errors. We found out that inactive connections are killed shortly after one hour, and for this reason we could not keep alive ssh connections longer without patching the GridWay ssh MAD.
- **Memory and network restriction in the user interface**. The GridWay metascheduler required a lot of memory to keep track of an enlarged pool with jobs connecting through ssh to cloud resources. We experienced unstabilities coming from saturated ethernet connections and large memory usage, causing the kernel to kill the gwd daemon due to "out of memory" on a stress proof. The user interface had 2GB of RAM and 1GB of swap, but regardless of the hardware specifications, a tuning of the configuration parameters is required to ensure full stability and permanent handling of the jobs.

6 Summary

A working model of a service manager in which grid resources are combined with HPC resources from Amazon Web Services has been presented in this paper. The restrictions regarding MPI and VO support were high, leading to a shortage on available resources from the grid.

To provide more power, Amazon instances have been customized and used on-demand. A novel component which interfaces with both, Amazon EC2 cloud and GridWay, allows for a utility usage after the saturation of grid resources but is limited to budget rate and system stability. The approach established has been a simple one and some work can be done in order to allow for more complex policies, such as prioritizing jobs or to use adaptive frameworks for minimizing expenditure. It is also mandatory to consider more machine types and providers. The use of a single type of instance (c1.xlarge) could be overcome with the recently announced Amazon "Cluster Compute Instances", capable to join into a single customizable larger cluster.

A real HPC application coming from the Xmipp package has been used in this context. The application provides image processing from microscopy and has been ported to the grid-cloud environment taking advantage of the parallel processing

thanks to the MPI-start suite. We compared a deterministic run of thousand images depending on the number of cores used and site. The results make noticeable some interesting conclusions: the application parallelization is quite good, as the execution time geometrically decreases with the number of cores used. However, typical real scenarios would use larger input data sets, one order of magnitude larger which require, in turn, to solve the inactive connection problems and better management of input storage.

Acknowledgements

This research was founded by the AWS in Education Research Grants program[9] by Amazon LTD., the GRID-CSIC project founded by the Spanish National Research Council-CSIC and by the Grant MAT2009-12364 from the CICYT (Comisión Interministerial de Ciencia y Tecnología, Spain).

We are grateful to Javier Fontán for the advice using the ssh plugin for GridWay and debugging. We would like to thank as well to the site administrators of the different Ibergrid sites: Tiago Sá from UMinho, Rui Ramos from UP, José Ramón Rodón from IAA-CSIC, Álvaro Fernández from IFIC-CSIC[10] regarding the support offered for the mpi jobs and VO duties. We are also indebted to the IFCA-CSIC staff and in particular to Enol Fernández, without whose support with the MPI-Start package this study could not have been carried out.

References

1. Gartner's Hype Cycle Special Report for 2010[11]. Gartner, Inc (5 August 2010).
2. Availability and Reliability monthly statistics, EGI document 402, February 2011.
3. E. Huedo, R.S. Montero, I.M. Llorente. Software - Practice and Experience 34 (7) 631-651 (2004)
4. C. Vázquez, E. Huedo, R.S. Montero, I.M. Llorente. Future Generation Computer Systems 27 (5), 600-605 (2011). doi:10.1016/j.future.2010.10.003
5. K. Dichev, S. Stork, R. Keller, and E. Fernández. Computing and Informatics, 27(3), 213–222 (2008)
6. J. Frank. New York: Oxford University Press (2006)
7. C.O.S. Sorzano, R. Marabini, J. Velazquez-Muriel, J.R. Bilbao-Castro, S.H.W. Scheres, J.M. Carazo, A. Pascual-Montano. J. Struct. Biol. 148(2), 194-204 (2004). doi:10.1016/j.jsb.2004.06.006
8. S.H.W. Scheres, M. Valle, R. Nuñez, R. Marabini, C.O.S. Sorzano, G.T. Herman, J.M. Carazo. J. Mol. Biol. 348(1), 139-149 (2005). doi:10.1016/j.jmb.2005.02.031

[9] `http://aws.amazon.com/education/aws-in-education-research-grants/`

[10] The IFIC-CSIC lcg-CE ce02.ific.uv.es was upgraded to CREAM during the research period.

[11] Web link: `http://www.gartner.com/DisplayDocument?doc_cd=205839`.